Journal of Intelligent Manufacturing manuscript No.
(will be inserted by the editor)

Learning and reuse of engineering ramp-up strategies for modular assembly systems

Daniele Scrimieri · Robert F. Oates · Svetan M. Ratchev

Received: date / Accepted: date

Abstract We present a decision-support framework for speeding up the ramp-up of modular assembly systems by learning from past experience. Bringing an assembly system to the expected level of productivity requires engineers performing mechanical adjustments and changes to the assembly process to improve the performance. This activity is time-consuming, knowledge-intensive and highly dependent on the skills of the engineers. Learning the ramp-up process has shown to be effective for making progress faster. Our approach consists in automatically capturing information about the changes made by an operator dealing with disturbances, relating them to the modular structure of the machine and evaluating the resulting system state by analysing sensor data. The feedback thus obtained on applied adaptations is used to derive recommendations in similar contexts. Recommendations are generated with a variant of the k-nearest neighbour algorithm through searching in a multidimensional space containing previous system states. Applications of the framework include knowledge transfer among operators and machines with overlapping structure and functionality. The application of our method in a case study is discussed.

Keywords Modular assembly systems · Ramp-up · Decision support · Learning · Classification

D. Scrimieri · S. M. Ratchev
Department of Mechanical, Materials and Manufacturing Engineering, University of Nottingham, Nottingham, NG7 2RD, UK
E-mail: {Daniele.Scrimieri,Svetan.Ratchev}@nottingham.ac.uk

R. F. Oates
ICOS Research Group, School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: Robert.Oates@nottingham.ac.uk

1 Introduction

Once the initial building of a complex assembly system has finished, the system is not immediately capable of producing at a high rate. There are frequent disruptions, the assembly process is not entirely understood and may be still subject to changes. As a consequence, the initial volume of production and quality of the final product are below the targets. The period between completion of development of a production system and full capacity utilisation is known as production ramp-up (Terwiesch and Bohn, 2001). Being able to shorten this phase and bring rapidly to the market high volumes of output whose quality has been assured would represent a considerable economic advantage for manufacturers. This is particularly important for advanced manufacturing systems, which require a high utilisation of the equipment in order to maximise the return on investments. Such systems are characterised by tight manufacturing plans, shorter product life cycles and increased amortisation of capital-intensive technologies (Almgren, 1999b).

In general, many sorts of problems in production systems can cause prolonged ramp-up times, of either technical or organisational nature: machines break down or do not operate efficiently, oversights on product design are discovered, personnel are not properly trained or do not have sufficient experience, there are delays in material supply (Almgren, 1999a; Terwiesch and Bohn, 2001). Ramp-up also takes place when a machine is reconfigured to react to market changes (Koren et al, 1999; Mehrabi et al, 2000) or is simply transferred from its manufacturer to the customer or a new plant.

In this paper, we focus on the engineering aspects of ramp-up concerning assembly systems and address the problem that ramp-up is not an automated process but heavily human-driven. As such, it requires detailed knowledge of the assembly technologies and processes involved, and it is

essentially carried out by trial and error, making a series of adjustments and evaluating their effects. Furthermore, the experience of an operator performing the ramp-up of a machine cannot be easily transferred to another operator for the ramp-up of the same type of machine, let alone a different one. In particular, the adjustments we consider are those regarding the control of assembly machines (e.g. locations of pick-and-place operations, pressure, cam angles, speed) and the geometry or design of the units involved (e.g. pallet geometry).

We present a decision-support framework for learning how to configure and tune assembly systems during ramp-up and reuse the acquired operational knowledge to speed up the ramp-up process itself. The general idea is that all adjustments made, whether they enhance the performance or not, bring new information about the system's evolution. Experience on adaptations in different system states is accumulated, where a state is represented by the status information gathered by the machine's sensors. Adaptations can be related to either the entire system or one single module and this information is modelled in the experience structure. If we can quantify the variation in efficiency then, when faced again with the same disruption, we can perform the adjustments that caused an improvement in the past and avoid those that had a negative impact. Such knowledge is related to the particular machine, product and process it was created on and can be utilised to provide decision support for the ramp-up of that production system. In addition, it can be employed on other machines with overlapping structure and functionality, and similar products or processes.

The ramp-up phase and the learning effects associated with it have been extensively studied from an economic perspective and many mathematical models have been developed to help control them. However, no computational framework has been proposed for learning and recommending engineering changes to equipment during ramp-up, apart from an outline of such a system (Oates et al, 2012).

The remainder of this paper is organised as follows. Section 2 illustrates typical problems occurring during ramp-up and what is meant by learning in manufacturing. Section 3 presents a methodology for carrying out the ramp-up process with the support of recommendations on adaptations. Section 4 describes how experience is structured and captured. Section 5 presents a technique for generating recommendations. In Section 6 a case study is discussed. Section 7 contains final remarks and conclusions.

2 Ramp-up and the role of learning

2.1 Characteristics of the ramp-up phase

As noticed by Surbier (2010), the "completion of development" as starting point of the ramp-up phase, indicated by Terwiesch and Bohn (2001), does not have well-defined boundaries. Although the development phase has finished, it is not rare to have late engineering changes throughout ramp-up or even production. These changes may be aimed at overcoming product and process design oversights. Winkler et al (2007) include in ramp-up a series of preparatory activities conducted in a pilot production phase, when prototypes or the final product are produced for testing or demonstration purposes in close-to-production conditions (pre series) or production conditions (pre production run). In this case, shortening ramp-up also means reducing such a preparatory phase.

Several characteristics of the ramp-up of production systems identified in the literature are summarised by Surbier (2010). Most of them are particularly relevant to assembly systems:

- The understanding of the process is poor when it starts. Knowledge is then accumulated through continuous learning, although learning does not occur systematically and with no difficulty;
- Cycle times are high and production capacity is lower than expected;
- There are numerous disturbances affecting process and product quality.

2.2 Critical events affecting progress

A wide range of events can occur during ramp-up, with either a positive or negative impact. Fjällström et al (2009) call them *critical events* and classify them in the following categories: product/quality, equipment/technique, process, suppliers/supply, personnel/education and organisation. These categories are similar to the four sources of disturbance identified by Almgren (1999a): product concept, material supply, production technology and personnel.

In the European research project FRAME,[1] four key types of events related to assembly systems and causing delays during ramp-up were identified:

- Unpredicted equipment failures (e.g. parameters out of tolerance due to either a slippage during production or a design oversight, communication infrastructure errors);
- Maintenance operations, which occur more often during ramp-up;
- Mistakes made by operators;
- Random events that require to restore the system to a previous operational state.

The industrial partners of the project, which were in the aerospace, automotive and medical devices sectors, faced these problems through a time-consuming and iterative process where process data were often manually collected and

[1] http://www.frame-eu.org

analysed. The aim of the project was to reduce the number of occurrences of these events and the time needed to deal with them through mechanisms for capturing information on how to handle them, generalising and reusing it on future occasions.

2.3 Learning by doing in production systems

Many managerial strategies are based on learning curves (Jaber, 2011), expressed in models whereby costs decline with rising cumulative output of production. Managerial strategies are intended in a general sense and include any decision aimed at increasing productivity, in relation to either the equipment or the personnel. In manufacturing, the more an organisation produces the more it learns and gains experience which facilitates progress. As a consequence, less time and lower costs for producing a unit are required. This is the so-called learn-by-doing. Learning curves are sometimes called interchangeably experience curves or progress functions, although major differences can be identified (Dutton et al, 1984). Such models have created a new understanding of production efficiency, which is a dynamic phenomenon and not a static one as it was assumed (i.e. solely dependent on the chosen combination of production elements) (Dutton et al, 1984).

Many factors can affect the way and rate an organisation learns and explain the differences in learning among different organisations. Argote and Epple (1990) analyse organisational forgetting, turnover and transfer of productivity gains. If knowledge is owned by individuals, turnover can cause loss of experience, especially in the case of lack of standardised procedures and when training new employees in a short time is not possible.

Transfer of knowledge across products and processes allows to be proficient in the manufacture of new products when these are related to old ones. If two products are similar and one has already been produced in the past while the other is going to be produced, the production of the new one can adapt the old process, reuse resources and utilise the experience acquired on the old one. This is clearly beneficial and makes ramp-up and learning of the new product faster. At the same cumulative production, the cost of the new product will be lower because of the higher accumulated knowledge achieved by the transfer from the other product. This difference will be reflected in the learning curve.

Argote and Ingram (2000) suggest that knowledge can be transferred by moving "knowledge reservoirs" and networks, that is, members of the organisation, tools, tasks and networks formed by combining them. This is, however, difficult to accomplish because of compatibility problems between networks. Alternatively, the knowledge reservoirs in the recipient unit can be modified through communication and training.

2.4 Learning by experimentation and learning strategies

In many studies of learning curves the only causal explanation of learning is considered to be the cumulative number of units produced. Thus, learning and progress seem to be achieved only by building up the cumulative volume of production. As a result, ramp-up looks like a manageable and predictable process, where little room is left for making decisions as pointed out by Terwiesch and Bohn (2001). These authors, instead, model learning in terms of experimentation and analyse the trade-off between short-term experimentation costs and long-term productivity increase. They demonstrate that early learning is more advantageous than late learning. In their work, experimentation is a form of induced learning, while production is seen as autonomous learning. Experiments are usually conducted in a series (Jaber, 2011, chap. 11). In a manufacturing process, for example, engineers may first try to isolate the cause of a problem, then test hypothesis explaining it and finally attempt to solve it.

In the context of the continuous improvement theory, Zangwill and Kantor (1998) criticise the approach of learning curves, which views learning as a "natural economic phenomenon". Learning curve theories can predict learning and how fast it occurs. However, they do not suggest practical procedures for increasing the rate of learning. Zangwill and Kantor argue that continuous improvement is activated by some form of learning and propose making learning happen in cycles. In a cycle, several strategies to make improvement are formulated and tested. If one appears to be effective it is adopted. The feedback obtained this way is used to define new strategies in the next cycle. Learning as an ongoing cyclic process is the basis of the organisational learning model of Argote and Miron-Spektor (2011), in which the organisational context affects the way experience is acquired and how learning occurs. New knowledge is then incorporated into the organisational context and affects future learning.

Terwiesch and Xu (2004) refer to learning during ramp-up in high-tech industry as the process of reducing the discrepancies between process specification and actual process execution. Once discrepancies have been eliminated, new improvement opportunities may arise (e.g. new technologies) and process modifications are needed. However, changes to the process make all the procedures established for the old process deprecated and new ones have to be laid down taking into account the changes. The authors discuss the "copy-exactly" ramp-up strategy, which consists in freezing the process recipe for a certain period of time during which learning can take place without being hindered by changes. The trade-off between long-term profit and short-term disruptions caused by process changes is also analysed by Carrillo and Gaimon (2000). Nembhard and Birge (1998)

discuss process adjustments during the ramp-up period of a continuous mix manufacturing process. Ngwenyama et al (2007) study the impact of software upgrades on productivity. Late upgrades cause a firm to miss the productivity gains offered by the new technology, while early upgrades require training and new routines.

Glock et al (2012) present a production-planning model for analysing the interdependencies between learning and growth in demand during the ramp-up of a manufacturing process, with the objective of minimising costs. This mathematical programming model indicates that, if finished products can be immediately shipped to customers, production plans should be synchronised to demand and stocks minimised. Synchronisation is accomplished by controlling the rate of learning or restructuring the workforce. The framework that we propose can help control learning through a better utilisation of the equipment and therefore provides a practical tool to apply the model of Glock et al (2012).

Through a case study in a major Swedish automotive company, Fjällström et al (2009) seek to find out what types of information are required to handle critical events in ramp-up, what the sources of information are, and how types and sources of information vary based on the category of the critical event and the experience of the personnel. The types of information they consider are problem, domain and problem-solving information. They conclude that problem and domain information are the most commonly needed and that domain information is not exclusively related to the class of the critical event being handled. For example, the domain information required to respond to a product-related critical event may include domain information on the equipment or process, in addition to the product itself. This suggests that a broad domain knowledge on product/process/equipment is necessary in order to handle events in any of these categories.

Organisations not only process information but also create knowledge, both tacit and explicit. Although knowledge is built by individuals, organisations play a crucial role in its creation process. Nonaka (1994) introduces a model of knowledge creation/conversion that explains how tacit or explicit knowledge creates new knowledge of the same type or is converted to the other type. The conversion of tacit knowledge into explicit knowledge is called "externalisation" and includes communication to articulate and elicit knowledge from, for example, customers or experts. The conversion of explicit knowledge into tacit knowledge is called "internalisation" and is a form of individual learning that involves absorbing explicit knowledge through practice and experience. The model focuses on the dynamic interaction between the two types of knowledge and their conversions, and has triggered subsequent research and debate (Nonaka and von Krogh, 2009). The methodology and computational framework that we describe in this paper aim at facilitating externalisation through automatic capture and formalisation of individual knowledge.

3 Methodology

Our general view is that in the ramp-up of an assembly system learning is accomplished through experimentation. With experiments, engineers test changes and try to eliminate disruptions. Learning is effectively the result of generalising the feedback obtained in each single experiment, building systematic strategies that can be adopted in the course of ramp-up.

3.1 Problems addressed and overall approach

Our methodology supports the decision-making process carried out by engineers during the ramp-up of an assembly machine. The aim is to reduce ramp-up time by automatically acquiring knowledge on the ramp-up process itself and reusing it in similar contexts to make informed choices on adaptations. As we will see in detail in the following sections, this is achieved through learning from the experience gained with experimentation. The methodology is meant to be implemented in a software framework, which would extend the functionality of assembly stations with the capability of learning from past experience.

The problems tackled by the methodology are twofold:

1. During ramp-up, the relationship between process parameters and performance of the assembly system is not known. Therefore an adaptive control cannot be applied. Ramp-up is conducted as a manual trial and error process, performing a series of changes to the parameters of the system and evaluating the impact, until a satisfactory configuration is found. It is not always evident during such a process whether the direction that is being followed leads towards the efficiency goals established. In fact, it may even bring the system into a non-functional state.
2. The ramp-up of an assembly machine is a process highly dependent on engineers' skills and experience, which requires knowledge not easily transferable among engineers and to new ramp-up problems.

Engineers resort to heuristic rules, domain knowledge and personal experience to adjust the system. Operational knowledge on adaptations could be formalised based on the domain of application. However, the domain may be too large to be analysed. On the contrary, experience can be formally defined and automatically captured (as it will be shown in Section 4). In addition, it arises from the joint application of heuristics and domain knowledge, so it implicitly includes them to some extent. Therefore, instead of try-

ing to find general adaptation rules, we can think of exploiting accumulated experience for finding successful adaptations.

A ramp-up process carried out without any external support boils down to the application of a sequence of changes followed by an assessment of the progress. Having past experience on particular adaptations before performing them would provide the engineer with insights into the effect of potential changes and support the decision-making process. The first step would be to distinguish adjustments that increase efficiency from those that do not. If detailed information on specific aspects of performance (e.g. cycle times, product quality) is available, a more refined application could consist in choosing adjustments based on a particular strategy that favours some elements of performance over others.

3.2 Use of recommendations

By (context-aware) recommendation we mean an atomic adaptation (i.e. a single adjustment) which is suggested in the current system state since it caused a positive result in a similar state in the past. Sections 4 and 5 describe, respectively, how to capture experience and use it for deriving recommendations on adaptations. Here we only discuss when recommendations are requested and how they are applied. We will call *strategy* a sequence of atomic adaptations.

Decision support in ramp-up is needed in two different but related cases:

1. The assembly system is not at the desired level of performance and the engineer decides to perform an adaptation to make improvement.
2. A disruption occurs, the performance drops and the engineer has to recover the previous functional state.

Case 1 Fig. 1 shows a high-level description of the sequence of steps executed during ramp-up with the support given by recommendations based on experience. The assembly system does not meet the efficiency goals and recommendations are requested to find a strategy for making progress towards the goals. There is no specific disruption to deal with. The available recommendations are dependent on the acquired experience and the similarity between the current system state and the states on which experience has been captured. At each step, recommendations are requested and if there are any, one is chosen, usually the one ranked first according to the adopted order. After applying it, the resulting system state is analysed and if the outcome is considered satisfactory, i.e. a performance gain has been achieved, the process can go to the next step. Otherwise, the previous state is restored and a different recommendation is followed, normally the next in the ranking. (It is assumed that changes

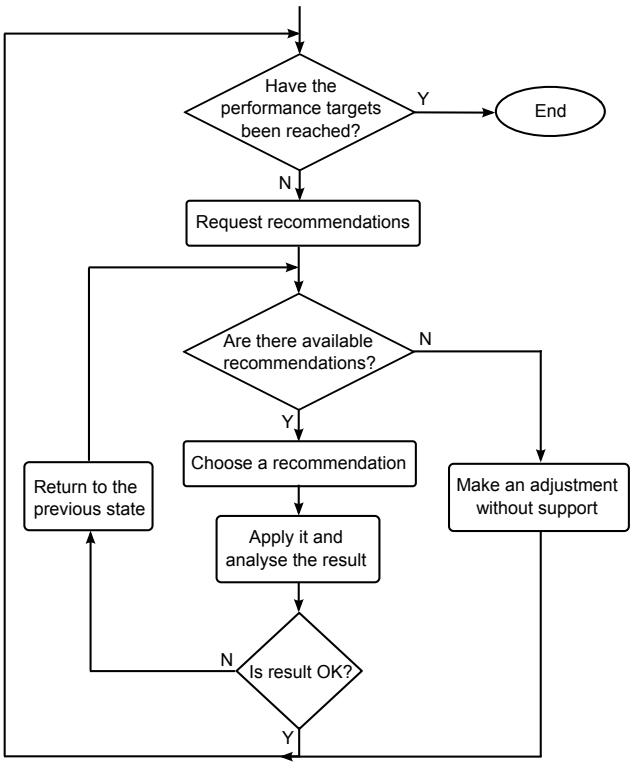

Fig. 1 Use of recommendations on adaptations to find a strategy to improve performance and reach the targets

are reversible, as it is usually the case.) If there are no (further) available recommendations, it is necessary to perform an adaptation autonomously. The process terminates when the performance targets have been reached.

Case 2 The case when a disruption has occurred is handled in a similar manner. Decision support is sought to devise a strategy to recover from the disruption. The procedure to follow is analogous to the one in Fig. 1, with the difference that the process ends when the disruption has been eliminated.

4 Experience capture

Experience capture is the process of acquiring information on adaptations being made, relating them to the modular structure of the machine and measuring their effect. Recommendations on executable adaptations are generated using the experience captured (as described in Section 5).

4.1 Structure and control of a modular assembly system

A modular assembly system can be described as a collection of processing units or modules assembling or checking parts. Each part is processed sequentially. The output of a module is given as input to the next module in the line.

Sub-assemblies of the same product can also be processed in parallel.

Each module can be controlled either separately or in conjunction with others. In the former case, a module has no access to information about the sub-processes carried out by the other modules or the overall assembly process. (It could, however, inspect the parts received to obtain such information.) In the latter case, modules share process data. A change in control affects all the controlled entities. Nevertheless, each individual module can additionally have a specific configuration which does not apply to the others. For example, consider an assembly system where all the modules are driven by the same motor and drive shaft but there are pick-and-place modules operating on different locations or having different pressures.

Modules can be replaced with others having similar functionality and interfaces in case of breakdown or changes to the process specification. Indeed, during ramp-up several modules might be replaced and tested. Modules are the smallest entities we are interested in for learning and when they are moved from one system to another we want to transfer also the knowledge accumulated about them. The analysis of the operation of a single module independently from the others does not normally provide us with all the information we need to measure its performance. This is evident in systems where the check of an assembly operation happens in a subsequent step by means of a dedicated module. In this case, we can measure the quality of the output of a module and assess its operation only when the check takes place.

Our object of study is given by systems providing intermediate quality checks on the parts being assembled and a last check on the final product. Within the scope of this paper, we use the following definitions:

Subsystem: a group of one or more modules performing assembly steps and a module checking the quality of the sub-assemblies at the end of these assembly steps;

System: a group of subsystems processing parts sequentially, preceded by a module checking the presence of parts and optionally followed by a further module checking the quality of the final product.

The process executed by a subsystem can be configured by setting subsystem-specific parameters in a *process recipe*. We assume the specification of a process recipe for each subsystem. In addition, there can be a process recipe for the entire system, which defines system-level parameters affecting more than one subsystem, possibly all. For instance, the layout of the pallets carried by a conveyor belt and processed by every subsystem is a system parameter specified in the system recipe. For each subsystem and the whole system a KPI (key performance indicator) is defined to measure the performance. In this paper, we only assume that a KPI function takes non-negative values and is such that the higher the value, the better the performance. Usually KPIs are functions of sub-assembly or assembly quality and throughput.

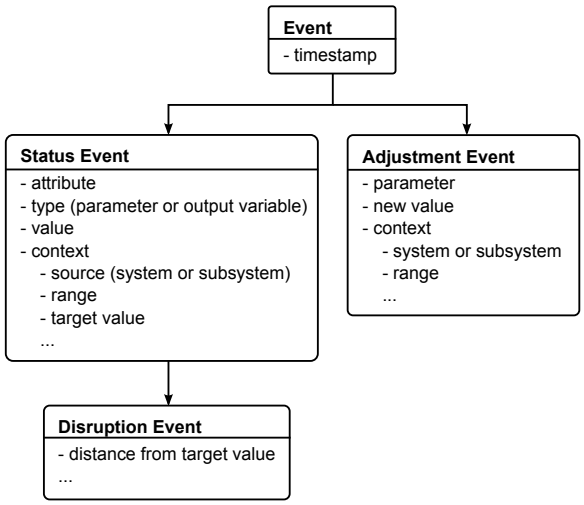

Fig. 2 Classification of event types

4.2 Generation of events and status information

We use assembly systems equipped with PLCs generating real-time signals which are then transformed into high-level data structures, handled by our software, that we call *events*. Events contain information on cycle times, results of checks or measurements, adjustments performed, modules added or removed, product item or pallet being processed, and anything else that can be monitored and is useful to characterise the system state or measure its performance. In fact, events are used to capture the state of the system at specific points and identify changes being made.

We differentiate two types of events: *status events*, conveying information about the current status of the system, and *adjustment events*, describing changes to the process recipe. In addition, we identify a sub-type of status events, called *disruptive events*, generated when a disruption occurs. In general, by disruption we mean a significant variation in a monitored variable. Disruptions can be detected by statistical process control or condition monitoring techniques. Usually they are symptoms of suboptimal conditions or developing failures and require human intervention.

Let us call *attribute* each process parameter and monitored variable whose detected values are transmitted by status events. Attributes can be relative to either individual subsystems or the entire system. Not all attributes are available all the time, since their presence is dependent on the modules and sensors installed at a specific moment. Fig. 2 summarises event types and their content.

We do not make assumptions on the frequency of generation of status events. Also, different variables can get up-

dated at different rates. However, it is required that the experience capture software be able to access the latest values of all attributes. This means that, if the information gathered is not up-to-date or incomplete, the software needs to wait for the generation of new events containing the latest values. In particular, the detection of a change involves the processing of status events generated after it. The effect of a change may not be immediately visible, though. For example, batch statistics on a subsystem are obtained when it has finished to process a batch. After a change, in order to get up-to-date statistics or calculate a KPI, it is necessary to process an entire new batch.

Associated with each attribute and adjustment there is a *context* which specifies to which module and subsystem they are related and provides the relevant domain information (e.g. range of values, type, unit of measurement, target value). Attributes and adjustments by themselves do not uniquely identify a module or subsystem. In fact, several modules or processes can be grouped into classes sharing the same set of attributes or adjustment types. For example, all pick-and-place modules in a workstation can have the same adjustment types for specifying the cam angles when grippers open or close. Additional module-specific attributes can also exist.

4.3 Representation and creation of experience

In our framework, experience is the representation of changes. It describes an experiment made and its outcome. Its creation is triggered by adjustment events. At the subsystem level, an experience instance is related to a change to a subsystem and includes the effect of the change on the subsystem. At the system level, experience is the result of either a subsystem or system change and includes the effect of the change on the whole system.

Suppose that we have a system consisting of m subsystems. For each $1 \leq i \leq m$, let $\text{Attr}_i = \{a_1, \ldots, a_{n_i}\}$ be the set of all attributes that can be used to characterise the state of subsystem i over a certain period of time, and let D^j ($1 \leq j \leq n_i$) be the domain of attribute a_j. Attr_i also includes all the system-level attributes which are relevant to subsystem i. We represent a state S of subsystem i at a certain point in time by a mapping from i's attributes to their respective values, i.e. by a function $S : \text{Attr}_i \to \bigcup_j D^j \cup \{null\}$ such that:

$$S(a_j) = \begin{cases} d \in D^j & \text{if attribute } a_j \text{ is present in state } S \\ null & \text{otherwise} \end{cases}$$

In addition, let D^S denote the set of attributes present in state S, i.e. all attributes a_j such that $S(a_j) \in D^j$. We denote the set of states of subsystem i by State_i. The KPI of subsystem i is a function defined on State_i.

A subsystem enters a new state every time any of its attributes changes or a module is added, removed or replaced. When a new module is inserted, the associated attributes become available, although not necessarily all at the same time. On the contrary, when a module is removed, the associated attributes are no longer available. A transition from S to T will be written as $S \to T$. During its operation, a subsystem enters a sequence of states $S_1 \to S_2 \to \cdots \to S_l$, where for each $1 \leq k \leq l-1$ there exists a j such that one of the following two conditions holds:

1. $a_j \in D^{S_k} \cap D^{S_{k+1}}$ and $S_k(a_j) \neq S_{k+1}(a_j)$ (i.e. a_j has changed value in the transition $S_k \to S_{k+1}$);
2. $a_j \in D^{S_k} \triangle D^{S_{k+1}}$ (i.e. a_j has been introduced or removed in the transition $S_k \to S_{k+1}$).

An adjustment is specified by a type, which indicates what the change is about (e.g. pressure, conveyor belt speed, cam angles, pick-and-place positions), and a value, which depends on the type. Let Adj_i be the set of adjustments for subsystem i. An adjustment $adj \in \text{Adj}_i$ has the form (t, v), where t is the type and v is the value.

Subsystem experience We can now define an experience instance *exp* for subsystem i as follows:

$$exp = (adj, S, C, S')$$

where

- $adj \in \text{Adj}_i$;
- $S \in \text{State}_i$ is the state when *adj* is made;
- $C \subset \text{Attr}_i$ is the set of attributes that caused a disruption in a certain period of time before performing *adj*;
- $S' \in \text{State}_i$ is the first state after *adj* has been performed and all available attributes have been updated.

The creation of *exp* can be illustrated by the following sequence of transitions, where $\xrightarrow{adj}$ denotes the transition when *adj* takes place:

$$\cdots \to S = S_0 \xrightarrow{adj} S_1 \to \cdots \to S_k = S'$$

where $1 \leq k \leq n_i + 1$ and $D^{S_1} \subset \cdots \subset D^{S_k}$. For each $1 \leq j \leq k$, one of the following conditions holds:

1. S_j contains an updated value for an attribute which is also in S;
2. S_j adds a new attribute which is not present in S (i.e. *adj* is about the addition of a module);
3. in case $j = 1$ and a module has been removed or replaced, S_1 is equal to S except for the absence of the attributes related to the module removed.

The existence of intermediate states between S and S' is due to the fact that attributes can get updated at different rates and, when a module is added, they may not all become available at the same time.

System experience Let SysAdj be the set of system-level adjustments and, for $1 \leq i \leq m$, let:

- $S_i \in \text{State}_i$ be the state in which subsystem i is when *adj* is made;
- $C \subset \text{Attr}_i$ be the set of attributes that caused a disruption in subsystem i in a certain period of time before performing *adj*;
- $S'_i \in \text{State}_i$ be the first state of subsystem i after *adj* has been performed and all available attributes have been updated.

A system-level experience instance *sysexp* has the following form:

$$sysexp = (adj, S, C, S')$$

where $adj \in \text{SysAdj} \cup (\bigcup_i \text{Adj}_i)$, $S = \bigcup_i S_i$ and $S' = \bigcup_i S'_i$. Note that system attributes are also specified (with the same value) in all subsystem states in which they are relevant. Analogously to subsystems, the system KPI is calculated on a system state, given by the union of the subsystem states.

5 Derivation of adaptation recommendations

We have presented a general technique for capturing and representing experience. What is still missing is an automatic procedure employing stored experience in a profitable context-aware way to systematically derive recommendations on adaptations and come up with a successful strategy. Upon request the engineer should be provided with a list of recommended adjustments, ranked from best to worst. They will then choose one as described in Section 3. Adjustments ranked last may deserve attention in order to avoid them.

The problem of finding an appropriate adjustment to perform in a specific system state can be seen as a *classification problem*. This is a typical problem in machine learning, where the system learns how to classify instances based on a given set of examples. Each example contains an input vector and a discrete output value that specifies the class of the example. The set of examples constitutes the *training set* for the learning system. The system receives the training set and tries to generalise the association between input vectors and output classes. The output of this process is a *classifier* program. When given a new instance not included in the training set, the classifier predicts its correct class.

A variety of learning methods have been developed for classification, such as artificial neural networks, instance-based learning, support vector machines and decision trees (Marsland, 2009). Applications in manufacturing include artificial neural networks for optimising a gas metal arc process (Lin, 2012) and for reliability assessment of structural systems (Patel and Choi, 2012), decision trees for container stacking strategies (Kang et al, 2006), k-nearest neighbour classifiers for tissue characterisation (Uchino et al, 2013) and text categorisation of customers' preferences (Li et al, 2009).

For the ramp-up problem and the experience model we have defined, the initial states and adjustment types of experience instances represent the training set, with the adjustment types being the classes. The instances to classify are the states, entered by the assembly machine throughout the ramp-up phase, in which the engineer requests support on adaptation. The classification process guides the engineer in finding an appropriate adjustment in each of these states. It is important not only to find the optimal setting of parameters, but also the sequence in which making changes as, in general, there could be constraints on applicable changes in a certain state.

5.1 A variant of the kNN algorithm

The technique that we present is a variant of the *K-nearest neighbour* algorithm (kNN) (Cover and Hart, 1967; Aha et al, 1991). In kNN, given a case to classify, the aim is to find the k nearest instances in the training set, minimising a certain distance measure.[2] The distance measure expresses the similarity between examples in the sense that the smaller the distance between two examples, the more similar they are supposed to be. The class to assign to an instance is the most common among the k neighbours or is chosen by adopting some voting scheme.

In addition to similarity between system states, our approach to classification uses performance estimates based on KPI values calculated on the experience. The resulting measure is not itself a distance function, though (i.e. a non-negative and symmetric function satisfying the triangle inequality). The method is equally applicable at both subsystem and system level.

5.2 Distance function

We define the distance $d(S_1, S_2)$ between system states S_1 and S_2 over attributes $a_1, \ldots, a_n$. Since we handle both numerical and categorical attributes, we use the *heterogeneous euclidean-overlap metric* (Wilson and Martinez, 1997):

$$d(S_1, S_2) = \sqrt{\sum_{i=1}^{n} (d_i(S_1, S_2))^2},$$

where $d_i(S_1, S_2)$, for $1 \leq i \leq n$, is the distance between S_1 and S_2 on attribute a_i:

$$d_i(S_1, S_2) = \begin{cases} 1 & S_1(a_i) = null \text{ or} \\ & S_2(a_i) = null, \\ \text{overlap}(S_1(a_i), S_2(a_i)) & a_i \text{ is nominal,} \\ \text{rndiff}_i(S_1(a_i), S_2(a_i)) & \text{otherwise.} \end{cases}$$

[2] k is usually a small odd integer.

The function overlap gives the value 0 if its arguments are the same, otherwise the value 1. The function rndiff_i (range normalised difference), calculated if attribute a_i is numerical, is defined as:

$$\text{rndiff}_i(x,y) = \frac{|x-y|}{\max(a_i) - \min(a_i)},$$

where $\max(a_i)$ and $\min(a_i)$ are, respectively, the maximum and minimum values of a_i observed in the training set.

Neighbours are found using the distance function, which calculates distances on all attributes. Redundant, irrelevant, interacting or noisy attributes have a negative impact on the similarity search. To deal with these issues, many weight-setting methods have been proposed. They are categorised and compared by Wettschereck et al (1997). A scheme tailored to our problem would assign higher weights to attributes that have caused a disruption either in the state where a recommendation is requested or in the experience. With such a scheme, states exhibiting the same disruption would be considered closer than those that do not.

5.3 Similarity-performance function

We are interested not only in finding the most similar experience instances, but also those containing the best adaptations, i.e. producing the highest KPI values. An experience instance with an initial state identical to the one we want to classify, but whose adjustment causes a performance drop would not be helpful. Based on this observation, we also analyse the KPI of the final state in the experience. Let $E = (adj, S, C, S')$ be an experience instance, T be a state and *kpi* be a KPI function. We define the *similarity-performance function* $e_{kpi}(E,T)$ as follows ($kpi(S') \neq 0$):

$$e_{kpi}(E,T) = \frac{d(S,T)}{kpi(S')}$$

If $kpi(S') = 0$, it means that *adj* made the system non-functional, so $e_{kpi}(E,T)$ is defined to be infinitely large in this case. Apart from the similarity between states, this function takes into account the performance according to the experience acquired. The smaller the value of this function, the better the adjustment of the experience is supposed to be in the state being examined.

5.4 Vote weighting

Given a state T in which recommendations are needed and a KPI function *kpi*, we search for the k experience instances E_i with the smallest values of $e_{kpi}(E,T)$. Once they have been found, a simple way of classifying T is to assign to it the most common class among the experience instances' classes. In kNN, the value k is usually chosen as a small, odd integer, e.g. 1, 3 or 5. Even numbers are not used to avoid ties that would have to be resolved by some other voting scheme. Higher values may reduce the effect of noise. A suitable k is typically chosen by cross-validation.

If $k > 1$, the problem with this approach is that the most frequent class in the training set is likely to appear in the majority of the selected k instances. Several solutions to this problem have been suggested. They normally take into account the distance from the neighbours to the instance to classify. Dudani (1976) uses the distance from the neighbours as weight in the voting, giving more influence to the closest ones. Other variations include radial basis function networks (Wasserman, 1993) and probabilistic neural networks (Specht, 1992), where all instances are weighted and contribute to the voting, albeit the distant ones have little influence. Thus, in this case there is no longer the need of choosing an appropriate k. García-Laencina et al (2008) use mutual information to weight the distance.

Following the idea of weighting votes with the respective distances, we propose the following weighting scheme that considers both distance and performance impact of adaptations, according to the similarity-performance function e_{kpi}. Let $E_1, \ldots, E_k$ be the k experience instances with the smallest values of $e_{kpi}(E,T)$, in ascending order. We assume that $e_{kpi}(E_k,T) \neq e_{kpi}(E_1,T)$ (in particular that $k \neq 1$) and assign a weight $w(E_i)$ to experience instance E_i, given by:

$$w(E_i) = \frac{e_{kpi}(E_k,T) - e_{kpi}(E_i,T)}{e_{kpi}(E_k,T) - e_{kpi}(E_1,T)}$$

For each class, the weights of its examples among $E_1, \ldots, E_k$ are summed up. Instance T is assigned the class with the largest sum of weights.

5.5 Construction and ranking of recommendations

Let T be the state in which recommendations are requested and *kpi* be a KPI function. Recommendations can be generated only if there exist experience instances with a final state having a KPI value greater than $kpi(T)$ and with an adjustment applicable in the current state (e.g. not already applied). We distinguish two cases:

$k = 1$ Suppose that $E' = (adj, S, C, S')$ is the experience instance that minimises $e_{kpi}(E,T)$. The recommended adjustment is *adj*, which specifies both the type of adjustment (i.e. the class to assign to T) and the value to set. The KPI being sought is $kpi(S')$. If there is more than one instance minimising $e_{kpi}(E,T)$, the one whose final state has the highest KPI is selected. Further recommendations can be generated based on other experience instances, in ascending order of e_{kpi}. They can be about other adjustment types or just different values.

$k > 1$ The recommended adjustment type t is given by the class selected over experience instances $E_1, \ldots, E_k$ using the voting scheme with the similarity-performance function e_{kpi}. The adjustment value is given by the weighted mean of the adjustment values of all experience instances among $E_1, \ldots, E_k$ with adjustment type t. The weights are the same ones used in the voting (i.e. the weight assigned to E_i's adjustment value is $w(E_i)$). Analogously, the KPI being sought is given by the weighted mean of the KPI values of the final states of all experience instances among $E_1, \ldots, E_k$ with adjustment type t. Further recommendations are given by the other classes of $E_1, \ldots, E_k$, in descending order of sum of weights.

6 Case study

We have applied our framework in the ramp-up of an assembly system for injection pens in the medical industry. An injection pen is composed of three parts: a body, a tank and a cap. Parts are fed in the line by pallets carried on a conveyor. Each pallet contains nests for holding the parts needed for assembling 3 pens, that is, 3 bodies, 3 tanks and 3 caps. The assembled pens are placed on the same pallet that initially contained their individual parts. The assembly machine is cam-driven and consists of 7 modules performing the following operations:

1. Check of the presence of all parts on the pallet;
2. Pick-and-place of tanks into bodies;
3. Check of the correct insertion of tanks in bodies;
4. Pick-and-place of caps on pens;
5. Check of the correct insertion of caps on pens;
6. Reversal of pens;
7. Check after reversal.

Each pick-and-place module operates 3 grippers, one for each pen to assemble. The machine is highly flexible, as all modules can be configured through a set of module-specific parameters based on the product to assemble. Fig. 3 shows a picture of the machine, where all modules are visible from left to right (except for module 3, which was not installed at that moment). Three subsystems can be identified:

- Subsystem 1 comprising modules 2 and 3;
- Subsystem 2 comprising modules 4 and 5;
- Subsystem 3 comprising modules 6 and 7.

For each subsystems, the KPI is given by the percentage of parts in a batch passing the subsystem's check. The system KPI is the percentage of good pens in a batch and coincides with the KPI of subsystem 3 .

Table 1 contains some system-level experience instances created during the ramp-up of this machine. They describe changes made to the closing angles of the grippers. Only a small subset of the attributes is shown. All other attributes have the same or very close values in those instances and in the machine states examined in the rest of this section.

We illustrate two examples of use of this small experience base, where we apply the search with $k = 1$. They are both related to adaptations made to the machine after a breakdown and the replacement of some mechanical components. They illustrate how to work out a strategy for changing the grippers' closing angles in two different cases. While in the first example recommendations are based on identical or very similar experience states and all adaptations are successful, in the second the similarity is lower and some recommended adaptations do not cause an improvement. The distance between states is calculated only on the attributes listed, which are the most relevant, and is thus an approximation. In particular, the distance (and therefore the e_{kpi} function) may be 0 for states which are very close but not identical. The range of the numerical attributes, used for normalising the distance, is 20.

Example 1 Table 2 shows the sequence of states the machine goes through by applying our method. Fig. 4 shows the recommendations provided at each step in a tree-like representation. Nodes represent machine states. Edges represent recommended changes. The labels associated with edges have the form *experience instance/similarity-performance*, where experience instance is an instance E from Table 1 and similarity-performance is given by $e_{kpi}(E,T)$, with T being the state from which the edge starts. The children of a node are the states that can be reached by applying the recommended changes indicated in the respective edges. Children are ordered from left to right by increasing values of e_{kpi} (in the figure a maximum of 3 children are shown). Nodes depicted with solid circles are the states actually entered by the machine and contain the relative KPI. Nodes depicted with dashed circles are states relative to recommendations not used. The root node is the initial state of the machine.

The machine is initially in state 1. We perform the following recommended adaptations:

1. The first recommended change is given by experience instance 2, at distance 0, which should take the KPI to 80.8%. It suggests that the angle of the cap gripper be changed to 320. We follow this recommendation and the machine enters state 2. The KPI increases to 80.2%.
2. At this point, the best adjustment is given by instance 6, again at distance 0. This adjustment is about changing the angle of the tank gripper to 320. With this change we achieve a better performance (82.5%) and the new state is state 3. Note that also instance 8 is at distance 0, but we choose instance 6 because of the slightly higher KPI.
3. The only relevant experience instance is 10. As a result of the recommended change (setting the cap gripper angle to 318), the machine enters state 4, with a KPI equal to 86.1%.

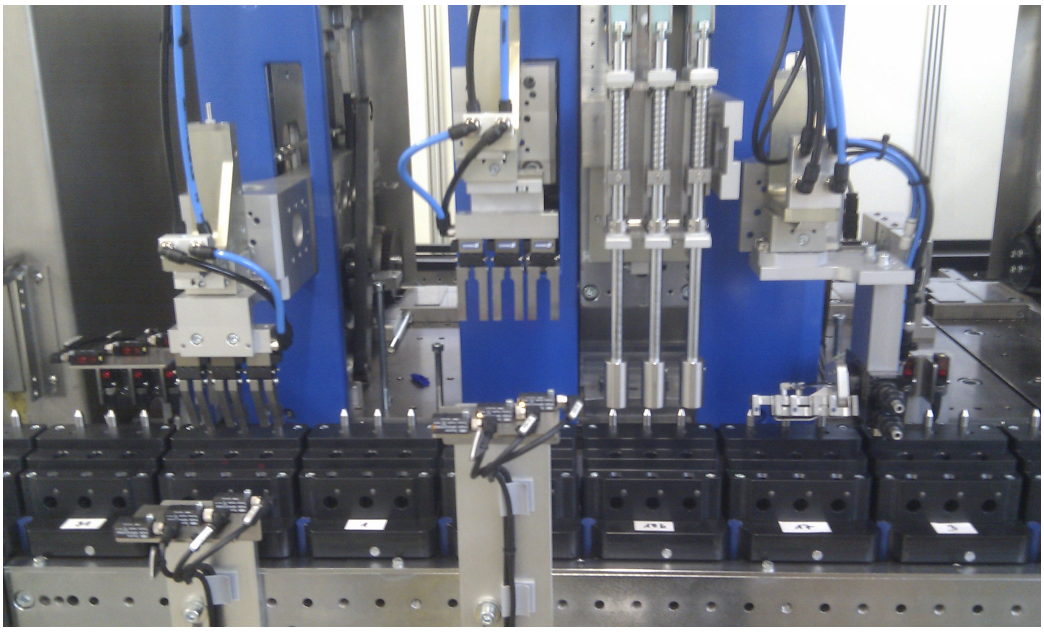

Fig. 3 The assembly machine for medical injection pens of the case study

Exp. instance	Tank gripper angle	Tank gripper type	Cap gripper angle	Reversal gripper angle	Good parts before change (%)	Adjustment	Good parts after change (%)
1	318	T1	322	318	79.3	set reversal gripper angle to 320	74.5
2	318	T1	322	318	78	set cap gripper angle to 320	80.8
3	318	T1	322	318	76.5	set tank gripper angle to 322	71.2
4	318	T1	322	320	72	set reversal gripper angle to 318	79
5	318	T1	322	320	75.4	set tank gripper angle to 320	77.4
6	318	T1	320	318	80	set tank gripper angle to 320	81.9
7	318	T1	320	318	77.5	set tank gripper angle to 322	73.5
8	318	T1	320	318	82.8	set cap gripper angle to 318	81.4
9	322	T1	320	318	70.3	set tank gripper angle to 320	82.4
10	320	T1	320	320	79.2	set cap gripper angle to 318	83.1
11	322	T1	320	318	75	set cap gripper angle to 318	75.6
12	318	T2	318	318	75.3	set cap gripper angle to 320	73.5
13	318	T2	318	318	72	set tank gripper angle to 322	80.8

Table 1 A small subset of the experience captured (showing only the attributes relevant to the examples discussed)

Now, there is no instance in Table 1 which could further increase the KPI, so the search stops. To sum up, we have got a KPI increase from an initial value of 76% to 86.1% in 3 steps.

Example 2 Table 3 shows the sequence of states the machine goes through by applying our method. Fig. 5 shows the recommendations provided at each step in the tree-like representation described above. The machine is initially in state 1. Note that the type of the tank gripper is T2, while in most of the experience instances it is T1. The two types are used in two different modules. Since they are very similar we can reuse the experience acquired on type T1 in this configuration. We perform the following recommended adaptations:

1. The first recommendation is based on experience instance 13, which is about changing the tank gripper angle to

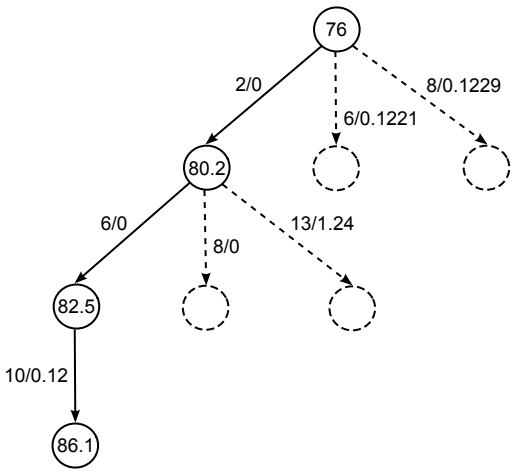

Fig. 4 Recommendations provided at each step in example 1

State	Tank gripper angle	Tank gripper type	Cap gripper angle	Reversal gripper angle	Good parts (%)	Exp. instance used in recommendation	e_{kpi}
1	318	T1	322	318	76	-	-
2	318	T1	320	318	80.2	2	0
3	320	T1	320	318	82.5	6	0
4	320	T1	318	318	86.1	10	0.12

Table 2 Sequence of states the machine goes through in example 1

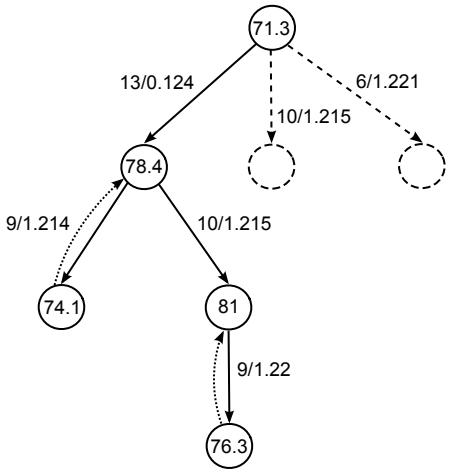

Fig. 5 Recommendations provided at each step in example 2

322. With this change, the machine enters state 2 and the KPI becomes 78.4%.
2. The next recommended adaptation derives from instance 9, which was generated using a T1 tank gripper. We apply it (setting the tank gripper angle to 320). The machine enters state 3 and the KPI drops to 74.1%. We then decide to backtrack to the previous configuration. The new state (state 4) is identical to state 2 except that the KPI calculated this time is slightly different (76.5%). Note that, if we had asked for recommendations, the same change that brings back to the previous configuration would have been suggested.
3. We opt for the next available recommendation (set the cap gripper angle to 318), given by instance 10. This adaptation is successful. The machine enters state 5 and the KPI increases to 81%.
4. The only applicable adaptation to try out is given again by instance 9. We apply it and the machine enters state 6 which is worse, so we backtrack to the previous configuration and the search stops.

To sum up, we have achieved a KPI increase from 71.3% to 80% in 6 steps (including backtracking), using experience generated on tank grippers of type T1 and T2.

7 Conclusions and discussion

We have presented a method for learning from experience captured on modular assembly systems. The recommendations generated guide the engineer throughout the ramp-up process. The use of a machine learning technique has two main advantages over other approaches to the construction of knowledge-based systems. First, there is no need for a knowledge engineer eliciting knowledge from shop floor operators, codifying it in a knowledge representation language and seamlessly integrating it in the knowledge base. In particular, the knowledge elicitation step presents several difficulties (Cooke (1994) reviews knowledge elicitation techniques, discussing their strengths and weaknesses). Second, in a traditional knowledge-based system, the knowledge includes a fixed set of patterns intended to cover all expected cases. A comprehensive knowledge base for ramp-up may be difficult or impractical to build because of the large amount of data to analyse or the intrinsic complexity of the domain. In contrast, a machine learning system does not require the manual construction of an explicit knowledge base. The knowledge, in this case, is automatically built directly from the data.

The similarity-performance function provides a simple yet powerful way of classifying system states and giving recommendations on adaptations by employing past experience on adjustments and performance measurements. There is no pre-built model of classification. The entire training set, represented by the experience captured, is stored and learning is delayed until classification time. For this reason, *k*-nearest neighbour algorithms are referred to as examples of *lazy learning*. During ramp-up, experience is accumulated until the targets on productivity have been reached. Afterwards, experience is still captured whenever changes are made to meet new demands or recover from disruptions. Therefore, the training set is constantly updated. Since a lazy learning algorithm does not require classification models to be constructed, there is no updating needed when new examples are added to the training set.

The accuracy of the technique depends on the similarity-performance function, and thus on the distance and KPI functions. The use of a distance function is based on the assumption that the more similar two states, the more likely an adjustment has the same effect on them. Without a weighting

State	Tank gripper angle	Tank gripper type	Cap gripper angle	Reversal gripper angle	Good parts (%)	Exp. instance used in recommendation	e_{kpi}
1	318	T2	320	318	71.3	-	-
2	322	T2	320	318	78.4	13	0.124
3	320	T2	320	318	74.1	9	1.214
4	322	T2	320	318	76.5	backtracking	-
5	322	T2	318	318	81	10	1.215
6	320	T2	318	318	76.3	9	1.22
7	322	T2	318	318	80	backtracking	-

Table 3 Sequence of states the machine goes through in example 2

scheme, all attributes equally affect the similarity measure. However, some attributes may be more important than others, in the sense that their similarity is more crucial in order to have the same adaptation effect. Furthermore, there may be interacting or noisy attributes. A review of weight-setting techniques to deal with these problems is presented by Wettschereck et al (1997).

Adaptation strategies are determined by searching for the best adaptation at each step, where the best adaptation is the one minimising the similarity-performance function. This is an example of *hill climbing* search: the adaptation found at each step is the best locally, i.e. considering only the neighbours, but it is not necessarily the best globally. This means that there could be a strategy bringing the machine in a better final state passing through intermediate states that do not minimise the similarity-performance function.

Although it has not happened in our experiments, a large experience base can result in high memory requirements and slow classification. Wilson and Martinez (2000) present some instance reduction algorithms that substantially reduce the memory required (also removing noisy instances), while maintaining generalisation accuracy. Samet (2006) discusses indexing techniques in multidimensional spaces and approximate searches to increase search speed. However, they may become less effective as the dimensionality grows (Beyer et al, 1999).

Oates et al (2012); Scrimieri and Ratchev (2013) describe some preliminary work on how to classify adaptations on assembly systems with different searches. Testing and reconfiguring by replacing modules is inspired by the plug and produce concept of Arai et al (2001), where an holonic system automatically recognises newly added devices. Further work in this direction includes the representation of devices' capabilities and physical requirements, and of assembly processes, in order to dynamically assign resources to processes and schedule operations. Evolvable assembly systems (Frei et al, 2007) have similar objectives.

Acknowledgements This research has been funded by the European Commission as part of the 7th Framework Program under the grant agreement CP-FP 229208-2, FRAME project.

References

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. *Machine Learning* 6(1):37–66

Almgren H (1999a) Pilot production and manufacturing start-up in the automotive industry - Principles for improved peformance. PhD thesis, Department of Operations Management and Work Organization, Chalmers University of Technology, Gothenburg, Sweden

Almgren H (1999b) Start-up of advanced manufacturing system - A case study. *Integrated Manufacturing Systems* 10(3):126–135

Arai T, Aiyama Y, Sugi M, Ota J (2001) Holonic assembly system with plug and produce. *Computers in Industry* 46:289–299

Argote L, Epple D (1990) Learning curves in manufacturing. *Science* 247(23):920–924

Argote L, Ingram P (2000) Knowledge transfer: A basis for competitive advantage in firms. *Organizational Behavior and Human Decision Processes* 82(1):150–169

Argote L, Miron-Spektor E (2011) Organizational learning: From experience to knowledge. *Organization Science* 22(5):1123–1137

Beyer KS, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbor" meaningful? In: Proceeding of the 7th International Conference on Database Theory

Carrillo JE, Gaimon C (2000) Improving manufacturing performance through process change and knowledge creation. *Management Science* 46(2):265–288

Cooke NJ (1994) Varieties of knowledge elicitation techniques. *International Journal of Human-Computer Studies* 41(6):801–849

Cover T, Hart P (1967) Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13(1):21–27

Dudani SA (1976) The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics* 6(4):325–327

Dutton JM, Thomas A, Butler JE (1984) The history of progress functions as a managerial technology. *Business History Review* 58(2):204–233

Fjällström S, Säfsten K, Harlin U, Stahre J (2009) Information enabling production ramp-up. *Journal of Manufacturing Technology Management* 20(2):178–196

Frei R, Ribeiro L, Barata J, Semere D (2007) Evolvable assembly systems: Towards user friendly manufacturing. In: Proceedings of the 2007 IEEE International Symposium on Assembly and Manufacturing

García-Laencina PJ, Sancho-Gómez J, Figueiras-Vidal AR, Verleysen M (2008) K-nearest neighbours based on mutual information for incomplete data classification. In: Proceedings of the 16th European Symposium on Artificial Neural Networks, pp 37–42

Glock CH, Jaber MY, Zolfaghari S (2012) Production planning for a ramp-up process with learning in production and growth in demand. *International Journal of Production Research* 50(20):5707–5718

Jaber MY (ed) (2011) *Learning Curves: Theory, Models, and Applications*. CRC Press

Kang J, Ryu K, Kim K (2006) Deriving stacking strategies for export containers with uncertain weight information. *Journal of Intelligent*

Manufacturing 17(4):399–410
Koren Y, Heisel U, Jovane F, Moriwaki T, Pritschow G, Ulsoy G, van Brussel H (1999) Reconfigurable manufacturing systems. *CIRP Annals - Manufacturing Technology* 48:527–540
Li X, Shi D, Charastrakul V, Zhou J (2009) Advanced P-Tree based K-Nearest neighbors for customer preference reasoning analysis. *Journal of Intelligent Manufacturing* 20(5):569–579
Lin HL (2012) The use of the Taguchi method with grey relational analysis and a neural network to optimize a novel GMA welding process. *Journal of Intelligent Manufacturing* 23(5):1671–1680
Marsland S (2009) *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC
Mehrabi M, Ulsoy A, Koren Y (2000) Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing* 11(4):403–419
Nembhard H, Birge J (1998) A startup procedure for process industries using a multiple objective nonlinear program. *IEE Transactions* 30(4):291–300
Ngwenyama O, Guergachi A, McLaren T (2007) Using the learning curve to maximize IT productivity: A decision analysis model for timing software upgrades. *International Journal of Production Economics* 105(2):524–535
Nonaka I (1994) A dynamic theory of organizational knowledge creation. *Organization Science* 5(1):14–37
Nonaka I, von Krogh G (2009) Tacit knowledge and knowledge conversion: Controversy and advancement in organizational knowledge creation theory. *Organization Science* 20(3):635–652
Oates RF, Scrimieri D, Ratchev S (2012) Accelerated ramp-up of assembly systems through self-learning. In: Proceedings of the 6th IFIP WG 5.5 International Precision Assembly Seminar, Springer, pp 175–182
Patel J, Choi SK (2012) An enhanced classification approach for reliability estimation of structural systems. *Journal of Intelligent Manufacturing* DOI 10.1007/s10845-012-0702-1
Samet H (2006) *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann
Scrimieri D, Ratchev S (2013) Capture and application of adaptation knowledge on assembly stations. In: Proceedings of the 11th IFAC Workshop on Intelligent Manufacturing Systems, pp 87–92
Specht DF (1992) Enhancements to probabilistic neural networks. In: Proceedings International Joint Conference on Neural Networks, vol 1, pp 761–768
Surbier L (2010) Problem and interface characterization during ramp-up in the low volume industry. PhD thesis, Institut polytechnique de Grenoble, France
Terwiesch C, Bohn RE (2001) Learning and process improvement during production ramp-up. *International Journal of Production Economics* 70:1–19
Terwiesch C, Xu Y (2004) The copy-exactly ramp-up strategy: trading-off learning with process change. *IEEE Transactions On Engineering Management* 51(1):70–84
Uchino E, Koga T, Misawa H, Suetake N (2013) Tissue characterization of coronary plaque by kNN classifier with fractal-based features of IVUS RF-signal. *Journal of Intelligent Manufacturing* DOI 10.1007/s10845-013-0793-3
Wasserman PD (1993) *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, New York, USA
Wettschereck D, Aha DW, Mohri T (1997) A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review* 11:273–314
Wilson DR, Martinez TR (1997) Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research* 6:1–34
Wilson DR, Martinez TR (2000) Reduction techniques for instance-based learning algorithms. *Machine Learning* 38:257–286
Winkler H, Heins M, Nyhuis P (2007) A controlling system based on cause-effect relationships for the ramp-up of production systems. *Production Engineering* 1(1):103–111
Zangwill WI, Kantor PB (1998) Toward a theory of continuous improvement and the learning curve. *Management Science* 44(7):910–920